**Algorithms and Data Structures (ADS) - COMP1819**

Develop and optimise solutions in Python with ADS and provide complexity analysis.

Group Name: 10-02

Team members:

|  |  |  |  |
| --- | --- | --- | --- |
| Member | Name | ID | Contribution % |
| 1 | Raut, Nihaar | 001309432 | 100% |
| 2 | Md. Nellikkaparambu, Md. Shuhaib | 001294977 | 100% |
| 3 | Lemoudden, Ilyass | 001355478 | 100% |
| 4 | Ali, Wais | 001324310 | 100% |
| 5 | Hernandez, Zak | 001355486 | 100% |
|  |  |  |  |

Contents

[1. Create unique solutions! 3](#_Toc158028396)

[**Student 1-6’s solution** 3](#_Toc158028397)

[**Results** 3](#_Toc158028398)

[2. Test and analyse your solution! 4](#_Toc158028399)

[**Your test cases:** 4](#_Toc158028400)

[**Running time graphs** 4](#_Toc158028401)

[**Complexity analysis** 4](#_Toc158028402)

[3. Optimise solutions! 4](#_Toc158028403)

[**Solution 1-2:** 5](#_Toc158028404)

[**Results** 5](#_Toc158028405)

[4. Compare the performance! 5](#_Toc158028406)

[**Time complexities and big-O notations** 6](#_Toc158028407)

[**Running time graphs** 6](#_Toc158028408)

[5. Reflecting on teamwork! 6](#_Toc158028409)

[**Contribution mark** 6](#_Toc158028410)

[**Limitation discussion** 7](#_Toc158028411)

[**Weekly journal** 7](#_Toc158028412)

[Reference 7](#_Toc158028413)

[Appendix A.1 - Proposed solution 1 - 6 8](#_Toc158028414)

[Appendix B - Test cases for correctness 8](#_Toc158028415)

[Appendix C - Evidence of team contribution 9](#_Toc158028416)

# Create unique solutions!

### **Student 1: Raut, Nihaar**

### **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Running time (s)** |
| 1 | 1, 2\_000 |  |  |
| 2 | 100, 10\_000 |  |  |
| 3 | 20\_000, 80\_000 |  |  |
| 4 | 100\_000, 2\_000\_000 |  |  |
| 5 | 2\_000\_000, 9\_000\_000 |  |  |
| 6 | 10\_000\_000, 100\_000\_000 |  |  |
| 7 | 100\_000\_000, 400\_000\_000 |  |  |
| 8 | 1\_100\_000\_000, 15\_000\_000\_000 |  |  |
| 9 | 15\_000\_000\_000, 100\_000\_000\_000 |  |  |
| 10 | 1, 1\_000\_000\_000\_000 |  |  |

### **Student 2: Nellikkaparambu Mohammed, Mohammed Shuhaib**

### **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Running time (s)** |
| 1 | 1, 2\_000 | Special numbers between 1 and 2000: [2, 3, 5, 797, 919, 929]  Total special numbers found: 20  Time taken: **0.00 seconds** | 0.00s |
| 2 | 100, 10\_000 | Special numbers between 100 and 10000: [101, 131, 151, 797, 919, 929]  Total special numbers found: 15  Time taken: **0.00 seconds** | 0.00s |
| 3 | 20\_000, 80\_000 | Special numbers between 20000 and 80000: [30103, 30203, 30403, 79397, 79697, 79997]  Total special numbers found: 48  Time taken: **0.04 seconds** | 0.04s |
| 4 | 100\_000, 2\_000\_000 | Special numbers between 100000 and 2000000: [1003001, 1008001, 1022201, 1993991, 1995991, 1998991]  Total special numbers found: 190  Time taken: **3.93 seconds** | 3.93s |
| 5 | 2\_000\_000, 9\_000\_000 | Special numbers between 2000000 and 9000000: [3001003, 3002003, 3007003, 7985897, 7987897, 7996997]  Total special numbers found: 327  Time taken: **29.13 seconds** | 29.13s |
| 6 | 10\_000\_000, 100\_000\_000 | Special numbers between 10000000 and 100000000: []  Total special numbers found: 0  Time taken: **1151.29 seconds** | 1151.29s |
| 7 | 100\_000\_000, 400\_000\_000 | NA | >3600s |
| 8 | 1\_100\_000\_000, 15\_000\_000\_000 | NA | >3600s |
| 9 | 15\_000\_000\_000, 100\_000\_000\_000 | NA | >3600s |
| 10 | 1, 1\_000\_000\_000\_000 | NA | >3600s |

### **Student 3: Lemoudden, Ilyass**

### **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Running time (s)** |
| 1 | 1, 2\_000 | Special numbers between 1 and 2000: [2, 3, 5, 797, 919, 929]  Total special numbers found: 20  Time taken: **0.00 seconds** | 0.00s |
| 2 | 100, 10\_000 | Special numbers between 100 and 10000: [101, 131, 151, 797, 919, 929]  Total special numbers found: 15  Time taken: **0.00 seconds** | 0.00s |
| 3 | 20\_000, 80\_000 | Special numbers between 20000 and 80000: [30103, 30203, 30403, 79397, 79697, 79997]  Total special numbers found: 48  Time taken: **0.04 seconds** | 0.04s |
| 4 | 100\_000, 2\_000\_000 | Special numbers between 100000 and 2000000: [1003001, 1008001, 1022201, 1993991, 1995991, 1998991]  Total special numbers found: 190  Time taken: **3.93 seconds** | 3.93s |
| 5 | 2\_000\_000, 9\_000\_000 | Special numbers between 2000000 and 9000000: [3001003, 3002003, 3007003, 7985897, 7987897, 7996997]  Total special numbers found: 327  Time taken: **29.13 seconds** | 29.13s |
| 6 | 10\_000\_000, 100\_000\_000 | Special numbers between 10000000 and 100000000: []  Total special numbers found: 0  Time taken: **1151.29 seconds** | 1151.29s |
| 7 | 100\_000\_000, 400\_000\_000 | NA | >3600s |
| 8 | 1\_100\_000\_000, 15\_000\_000\_000 | NA | >3600s |
| 9 | 15\_000\_000\_000, 100\_000\_000\_000 | NA | >3600s |
| 10 | 1, 1\_000\_000\_000\_000 | NA | >3600s |

### **Student 4: Ali, Wais**

The proposed python code is an interesting tool for finding numbers that are both prime and palindromic. To do this, it consists of multiple finely crafted components. The function that determines whether a given number is prime is the first one primic(); it uses effective algorithm to handle a variety of scenarios and maximise efficiency. In a similar vein, another function checks whether a given number reads the same both forward and backward is the palindromic(). The script also includes a way to find these unique numbers within the given range, which simplifies the process and eliminates needless calculations. The code shows strong performance and fast results when handling ranges from 1 to hundreds of millions. However, as it gets closer to very large ranges, like a billion or a trillion, its efficiency starts to decrease. The fundamental complexity of prime number testing at such large scales is the cause of this slowdown. Even with the optimised algorithms included in the code, there are just so many numbers to test in these large ranges, which naturally takes more time and computational power. Nonetheless, the script is still a useful computational and educational tool, especially useful for showing the ideas of prime numbers and palindromes in large but manageable numerical ranges.

1. **def** primic(num):
2. **if** num < 2:
3. **return** False
4. **if** num **==** 2:
5. **return** True
6. **if** num **%** 2 **==** 0:
7. **return** False
8. **for** i **in** range(3, int(num **\*\*** 0.5) **+** 1, 2):
9. **if** num **%** i **==** 0:
10. **return** False
11. **return** True

14. **def** palindromic(num):
15. **return** str(num) **==** str(num)[::**-**1]

18. **def** fetch\_special\_numbers(m, n):
19. special\_numbers **=** []
20. **for** number **in** range(max(2, m), n **+** 1):
21. **if** primic(number) **and** palindromic(number):
22. special\_numbers.append(number)
23. **return** special\_numbers

### **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Running time (s)** |
| 1 | 1, 2\_000 | Special numbers between 1 and 2000: [2, 3, 5, 797, 919, 929]  Total special numbers found: 20  Time taken: **0.00 seconds** | 0.00s |
| 2 | 100, 10\_000 | Special numbers between 100 and 10000: [101, 131, 151, 797, 919, 929]  Total special numbers found: 15  Time taken: **0.00 seconds** | 0.00s |
| 3 | 20\_000, 80\_000 | Special numbers between 20000 and 80000: [30103, 30203, 30403, 79397, 79697, 79997]  Total special numbers found: 48  Time taken: **0.04 seconds** | 0.04s |
| 4 | 100\_000, 2\_000\_000 | Special numbers between 100000 and 2000000: [1003001, 1008001, 1022201, 1993991, 1995991, 1998991]  Total special numbers found: 190  Time taken: **3.93 seconds** | 3.93s |
| 5 | 2\_000\_000, 9\_000\_000 | Special numbers between 2000000 and 9000000: [3001003, 3002003, 3007003, 7985897, 7987897, 7996997]  Total special numbers found: 327  Time taken: **29.13 seconds** | 29.13s |
| 6 | 10\_000\_000, 100\_000\_000 | Special numbers between 10000000 and 100000000: []  Total special numbers found: 0  Time taken: **1151.29 seconds** | 1151.29s |
| 7 | 100\_000\_000, 400\_000\_000 | NA | >3600s |
| 8 | 1\_100\_000\_000, 15\_000\_000\_000 | NA | >3600s |
| 9 | 15\_000\_000\_000, 100\_000\_000\_000 | NA | >3600s |
| 10 | 1, 1\_000\_000\_000\_000 | NA | >3600s |

### **Student 5: Hernandez, Zak**

### **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Running time (s)** |
| 1 | 1, 2\_000 | Special numbers between 1 and 2000: [2, 3, 5, 797, 919, 929]  Total special numbers found: 20  Time taken: **0.00 seconds** | 0.00s |
| 2 | 100, 10\_000 | Special numbers between 100 and 10000: [101, 131, 151, 797, 919, 929]  Total special numbers found: 15  Time taken: **0.00 seconds** | 0.00s |
| 3 | 20\_000, 80\_000 | Special numbers between 20000 and 80000: [30103, 30203, 30403, 79397, 79697, 79997]  Total special numbers found: 48  Time taken: **0.04 seconds** | 0.04s |
| 4 | 100\_000, 2\_000\_000 | Special numbers between 100000 and 2000000: [1003001, 1008001, 1022201, 1993991, 1995991, 1998991]  Total special numbers found: 190  Time taken: **3.93 seconds** | 3.93s |
| 5 | 2\_000\_000, 9\_000\_000 | Special numbers between 2000000 and 9000000: [3001003, 3002003, 3007003, 7985897, 7987897, 7996997]  Total special numbers found: 327  Time taken: **29.13 seconds** | 29.13s |
| 6 | 10\_000\_000, 100\_000\_000 | Special numbers between 10000000 and 100000000: []  Total special numbers found: 0  Time taken: **1151.29 seconds** | 1151.29s |
| 7 | 100\_000\_000, 400\_000\_000 | NA | >3600s |
| 8 | 1\_100\_000\_000, 15\_000\_000\_000 | NA | >3600s |
| 9 | 15\_000\_000\_000, 100\_000\_000\_000 | NA | >3600s |
| 10 | 1, 1\_000\_000\_000\_000 | NA | >3600s |

# Test and analyse your solution!

(Come up with test cases that demonstrate the correctness of your solution.

Explain why you chose these particular test cases.

Also, perform a complexity analysis to understand how efficient your solution is. You can present your results in a table, showing different inputs in the test cases along with their runtimes. Your group can create a graph to visualise how the runtime changes with different inputs.)

### **Your test cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **c** | **Input** | **Output** | **Justification** | **Student X results** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
|  |  |  |  |  |

### **Running time graphs**

A graph with red and blue lines

Description automatically generated

### **Complexity analysis**

# Optimise solutions!

(Choose two of the solutions you created earlier in (I) and improve their efficiency. Share your reasons for all the optimising steps that your group took.

Include the final code with outputs and running time measurements in the report’s Appendix section and also upload it for Deliverable 2.

Remember, your optimisation goal is to solve correctly as many test cases as possible, as shown in the test case table above.)

### **Solution 1-2:**

Which ones did you group choose and give reasons for all the optimising steps that your group took.

Short description and highlights of the improvement in your code, and the full code in the Appendix.

1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. queue = Queue()
3. queue.put(1)
4. queue.put(2)
5. queue.put(3)
6. queue.put(4)
7. queue.put(5)

10. reverseqQueue(queue) #your implementation
11. printQueue(queue)

### **Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Correctness** | **Running time (s)** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
|  |  |  |  |  |

# Compare the performance!

(Compare the performance of the two optimised solutions you created (III).

Focus on understanding their time complexities and big-O notations.

Create a graph to visualise how their running time compares to the test cases. Make sure to include detailed outputs, running time measurements for as many test cases as possible, and a graphical representation of your results in your submission.)

### **Time complexities and big-O notations**

### **Running time graphs**

A graph with red and blue lines

Description automatically generated

# Reflecting on teamwork!

(Your task is to write about the limitations you faced while working together and give each team member a contribution mark that everyone agrees on.

Keep a weekly journal, noting down communication logs and the earned credits for each member. Summarise what each member did during the project.

Make sure your whole report is well-organised, uses clear language, and includes proper references.

This will help us understand your teamwork better!

Please note that this coursework should take an average student who is up-to-date with tutorial work **approximately** **25 hours for each member** of the group

There is a group mark decided by the marker. Each team member's overall contribution is assessed on a scale from 0% to 100%, with agreement from the team. For instance, if a member did not contribute to problem optimisation, they might receive 80% out of the 100%. An 80% individual effort could result in 80% of the group mark, but the final decision rests with the marker.

)

### **Contribution mark**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | ID | Task 1 (30%) | Task 2 (20%) | Task 3 (20%) | Task 4 (15%) | Task 5 (15%) | **Contribution mark**  **(100%)** |
| Last, First (Group leader) | 001234561 | 30% | 20% | 20% | 15% | 15% | 100% |
| Last, First (Champion 1) | 001234562 | 30% | 20% | 20% | 15% | 15% | 100% |
| Last, First (Champion 2) | 001234563 | 30% | 20% | 20% | 15% | 15% | 100% |
| Last, First (Role) | 001234564 | 20% | 20% | 10% | 0% | 10% | 60% |
| Last, First (Role) | 001234565 | 20% | 20% | 0% | 0% | 10% | 50% |
| Last, First (Role) | 001234566 | 0% | 0% | 0% | 0% | 0% | 0% |

### **Limitation discussion**

Your group might discuss the technical challenges, participation/engagement, collaboration, leadership, problem-solving skills, creativity and innovation, or communication dynamic topics.

### **Weekly journal**

|  |  |  |
| --- | --- | --- |
|  | **Task note** | **Status** |
| **Week 1: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Champion 1) |  |  |
| Last, First (Champion 2) |  |  |
| Last, First (Role) |  |  |
| Last, First (Role) |  |  |
| Last, First (Role) |  |  |
| **Week 2: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Champion 1) |  |  |
| Last, First (Champion 2) |  |  |
| Last, First (Role) |  |  |
| Last, First (Role) |  |  |
| Last, First (Role) |  |  |
| … |  |  |

# Reference

Tuan Vuong, COMP1819ADS, (2022), GitHub repository, <https://github.com/vptuan/COMP1819ADS>

# Appendix A.1 - Proposed solution 1 - 6

You can try to use Pycharm or VSCode to paste Python code into Word document. Note that it is important to keep the Python code in good structure, and text format for readability.

1. """
2. This video has NO Sound
4. Spyder Editor: Spyder 4.2.1
6. This demo is for Lab 02 - Ex1 MinMax function
7. """
8. **import** time
10. **def** minmax(sequence):
11. min = max = sequence[0] # assuming no-empty
12. **for** val **in** sequence:
13. **if** (val > max):
14. max = val
15. **if** (val < min):
16. min = val
17. **return** (min,max)
19. #print(minmax([1,2,3,5]))

22. **def** measure\_time(input\_size):
23. sequence = [i **for** i **in** range(input\_size)] # input = a list [0,1,2,...]
24. #print(sequence)
25. start = time.time() # start timer
26. **print**(minmax(sequence)) # execute the function with the sequence
27. **print**("Input size=", input\_size, " Time taken=", time.time()-start)

30. # Now, we make input size larger, 2k, 10k,50k, 200k,1000k
32. k = 1000;
33. measure\_time(2\*k)
34. measure\_time(10\*k)
35. measure\_time(50\*k)
36. measure\_time(200\*k)
37. measure\_time(1000\*k)
39. # Now, we plot in Excel. The plot looks linear? This is O(n) because
40. # the for loop in line 12.

# Appendix B - Test cases for correctness

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Input** | **Output** | **Comments** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |

# Appendix C - Evidence of team contribution

Communication logs